

This application is submitted in the names of inventor Steve W. Brown, assignor to Zayante, Inc., a California Corporation.

5

SPECIFICATION

10

A METHOD AND APPARATUS FOR DYNAMIC LINK DRIVER CONFIGURATION

15

20

BACKGROUND OF THE INVENTION

1. Field of the Invention

25

This invention pertains generally to link driver configuration for IEEE Standard 1394 nodes. More particularly, the invention is a method for dynamic link driver configuration of link driver architecture for IEEE Standard 1394 modules.

30

2. The Prior Art

35

The Institute of Electrical and Electronics Engineers, Inc. (IEEE) defines the IEEE Standard 1394-1995 serial bus architecture in the document "IEEE Standard for a High Performance Serial Bus" published August 30, 1996 which is incorporated herein by reference. In IEEE 1394, the serial bus architecture is

defined in terms of nodes. In general, a node is an addressable entity (i.e., a logical entity with a unique address), which can be independently reset and identified. More than one node may reside on a single module, and more than one unit may reside in a single node.

5

A module is a physical device, comprising one or more nodes that share a physical interface. The address space provided by a node can be directly mapped to one or more units. A unit is a logical entity, such as a disk controller, which corresponds to unique I/O (input/output) driver software. On a multifunction
10 node, for example, a processor and I/O interfaces could be different units on the same node.

During initialization (startup) of a module, certain hardware devices of the module are checked and appropriate drivers are loaded as is known in the art. For
15 example, FIG. 1 illustrates a typical module device 1 having first and second nodes 2a, 2b. Nodes 2a, 2b include respective link layer services (LINK) 3a, 3b and physical layer services (PHY) 4a, 4b. During the start up of module 1, link driver 5a is loaded to configure LINK 3a, and link driver 5b is loaded to configure LINK 3b. The link drivers 5a, 5b provide necessary configuration information
20 which allows the LINKs 3a, 3b to carry out its link layer services. The prior art implementation provides a static link driver which is configured identically for each node 2a, 2b without regard for the type of communication that will be carried out by the node. Thus, link driver 5a is configured the same way as 5b, even though node 2a may carry out different communication than node 2b.

25

The prior art implementation of providing a static configuration for link drivers is not always optimal. For example, in IEEE 1394 communication, nodes may carryout asynchronous and isochronous communication. In asynchronous communication such as SBP (serial bus protocol), it would be advantageous to have a link device configured for data pumping to provide optimum performance for such asynchronous transfers. On the other hand, in isochronous communication such as AV/C (audio/video control), no advantage is provided if the link device is configured for data pumping, since AV/C commands do not have high bandwidth requirements. Rather, AV/C communication would benefit if the link device is configured for transferring isochronous data. Thus, the current implementation of providing a static configuration link driver for all LINKS (3a, 3b, for example) is a disadvantage.

As an example, LINK 3a may be able to receive and process 2054 byte packet sizes, while LINK 3b may be able to receive and process only 512 byte packet sizes. The prior art implementation would configure link device drivers 5a, 5b to handle 512 byte packet sizes, even though LINK 3a is able to handle larger (2054 byte) sizes. Thus, the capabilities of LINK 3a are not fully utilized where static link driver configurations are provided as is carried out in the prior art.

Accordingly, there is a need for a method which provides multiple link device driver configurations based on the capabilities of the link device and based on the specific behaviors of the link device. The present invention satisfies these needs, as well as others, and generally overcomes the deficiencies found in the background art.

An object of the invention is to provide a method for configuring link device drivers which overcomes the deficiencies of the prior art.

5 Another object of the invention is to provide a method for configuring link device drivers based on the capabilities of the link device.

Another object of the invention is to provide a method for configuring link device drivers based on the specific behaviors of the link device.

10

Another object of the invention is to provide a method for configuring link device drivers by ascertaining the capabilities of link devices before providing the configuration information for the link device drivers.

15 Further objects and advantages of the invention will be brought out in the following portions of the specification, wherein the detailed description is for the purpose of fully disclosing the preferred embodiment of the invention without placing limitations thereon.

20

BRIEF DESCRIPTION OF THE INVENTION

The present invention is a method and apparatus embodied in transaction layer software suitable for use with serial bus devices, such as IEEE standard 1394 serial bus devices for supporting multiple link device drivers. In its most
25 general terms, the invention acquires or otherwise ascertains the capabilities of

link devices and provides link device driver configurations to such link devices based on the link device's capabilities and behaviors, among other factors.

The invention further relates to machine readable media on which are
5 stored embodiments of the present invention. It is contemplated that any media suitable for retrieving instructions is within the scope of the present invention. By way of example, such media may take the form of magnetic, optical, or semiconductor media. The invention also relates to data structures that contain embodiments of the present invention, and to the transmission of data structures
10 containing embodiments of the present invention. The method and operation of the invention may be carried out by a conventional processor within the serial bus device as is known in the art.

In general the invention may be used to configure one or more link device
15 drivers in a module of a serial bus device according to the capabilities of the link devices, the behavior of the link devices, and other criteria.

In a first system embodiment, the invention operating in the transaction layer of the module is operatively coupled for communication to one or more link
20 devices for the configuration of the link device drivers for the link devices. In a second system embodiment, the invention operating in the transaction layer of the module is operatively coupled for communication to a device driver service which is operatively coupled for communication with one or more link devices. The device driver service provides messaging between the transaction layer and

the link devices for the configuration of the link device drivers. In general, the invention and the link devices communicate via driver control commands.

Sub B1 ~~The link devices which for which the invention provides link driver~~
 5 configuration may comprise the same or different types. In cases where the link devices are the same type, the invention configures the link driver for each link device according to the behavior of the link device (e.g., the type of communication carried out by the link device) as well as the capabilities of the link device. For example, in a module having first and second nodes, each node
 10 having a link device. The module may be configured such that the first node carries out asynchronous communication, while the second module may be configured for isochronous communication. Even though the link device in the first module is identical to the link device in the second module, the invention may provide link driver configuration optimized for asynchronous data transfer to
 15 the link device in the first module and link driver configuration optimized for isochronous data transfer to the link device in the second module to thereby
~~support the behavior carried out by each respective module.~~

Other criteria may be used to configure link devices including, for example,
 20 user defined input criteria provided by a user of the module.

In operation, during initialization, link drivers are "installed" or loaded according to the type of system involved. For embedded systems, the method for installing device drivers will vary depending on the needs of the implementation.
 25 Device drivers for locally resident drivers may be pre-compiled into a ROM image.

Under this arrangement, at boot time the drivers would be called to perform initialization thereof.

The transaction layer software of the present invention then queries each
5 of the link drivers to ascertain each link device's capabilities via a driver control
command. In the preferred embodiment, the transaction layer requests the link's
capabilities as soon as it becomes aware of the link device. In response, the link
drivers transmit its respective capabilities to the transaction layer software. The
capabilities of the link device may be statically provided in a resident storage
10 device, such as a BIOS (basic input/output system), for example. The transaction
layer software may receive additional configuration data such as user-defined
configuration, which may define specific behaviors of the module. The
transaction layer software then generates link driver configuration data according
to the link device capabilities and the other configuration data for each link
15 device. The generated link driver configuration is then transmitted to the
respective link device driver and loaded therein.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be more fully understood by reference to the
following drawings, which are for illustrative purposes only.

25 FIG. 1 is a functional block diagram of serial device module which carries
out device driver configuration according to the prior art.

FIG. 2 is a functional block diagram of an illustrative link data structure suitable for use with the present invention.

5 FIG. 3 is a functional block diagram of a first embodiment serial device module which carries out device driver configuration according to the present invention.

10 FIG. 4 is a functional block diagram of a second embodiment serial device module which carries out device driver configuration according to the present invention.

15 FIG. 5 is a flow chart showing generally acts for configuring link drivers according to the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

20 Persons of ordinary skill in the art will realize that the following description of the present invention is illustrative only and not in any way limiting. Other embodiments of the invention will readily suggest themselves to such skilled persons having the benefit of this disclosure.

25 Referring more specifically to the drawings, for illustrative purposes the present invention is embodied in the apparatus shown FIG. 2 through FIG. 4 and the method outlined in FIG. 5. It will be appreciated that the apparatus may vary

as to configuration and as to details of the parts, and that the method may vary as to details and the order of the acts, without departing from the basic concepts as disclosed herein. The invention is disclosed generally in terms of a method and system which provides multiple link device driver configurations based on the capabilities of the link device and based on the specific behaviors of the link device, although numerous other uses for the invention will suggest themselves to persons of ordinary skill in the art.

Referring first to FIG. 2, there is depicted a block diagram of an illustrative link data structure 10 suitable for use with the present invention. It will be appreciated that the data structure depicted in FIG. 2 and described herein is only exemplary and that other data structures may be used in conjunction with the present invention, such as database tables and trees, for example.

Link data structure 10 is a link list of link device nodes 12a through 12n, and is maintained by the transaction software (TNF kernel) of the present invention. Each link device node 12a through 12n represents a link device in the module and includes information about the link device. For example, in a module having two link devices, the first link device would be represented by node 12a and the second link device would be represented by node 12b.

Sub B2 ~~A main pointer 14 provides a link to the link data structure 10. Link devices nodes 12a through 12n further may include a peer pointer 16a through 16(n-1) to thereby allow TNF kernel to navigate the link data structure 10 for each link device of the module. Thus, node 12a includes a peer pointer 16a to~~

node 12b, node 12b includes a peer pointer to 12c, etc. Since the last node 12n does not have additional peers to point to, node 12n does not include a peer pointer, shown as a null pointer 18.

5 Each node 12a through 12n further includes a corresponding link configuration data structure 22a through 22n and a corresponding link capabilities data structure 24a through 24n. The link configuration data structure 22a through 22n is used by the TNF kernel to instruct the link device how it should be configured. For example, the link configuration data structure 22a
10 through 22n controls how the link behaves on the serial bus, among other things. The link capabilities data structure 24a through 24n provides the TNF kernel with information about the link device. For example, the link capabilities data structure 24a through 24n may provide such information about the corresponding link device as the link device's maximum sync packet capabilities,
15 support or non-support for busy mode, cycle accuracy, capability for cycle master, CSR-ROM support, and other like capabilities.

Referring now to FIG. 3, there is shown a functional block diagram of a first exemplary embodiment serial device module 26 which carries out device
20 driver configuration according to the present invention. Module 26 includes three nodes 28a through 28c, each having a respective link layer (LINK) device 30a through 30c connected to a respective physical layer (PHY) devices 32a through 32c. LINKS 30a through 30c provide the link services for the module 26 as is known in the art, and PHY devices 32a through 32c provide the physical
25 layer services for the module 26 as is known in the art. Each PHY device 32a

through 32n is connected to serial bus 34 through a conventional serial interface connection.

The module 26 further includes one or more unit architectures 36 to
 5 present to other devices on the serial bus. Unit architectures 36 may comprise conventional units, such as a disk controller or some other storage device and a scanner controller, for example.

The unit architectures 36 and the LINKS 30a through 30c are operatively
 10 coupled for communication to TNF kernel 38. The TNF kernel 38 provides transactional services for module 26 and the method of the invention as described herein and in further detail in conjunction with FIG. 5.

sub B3
 15 ~~In operation, when module 26 is initialized, Link drivers 40a through 40c~~
 are loaded (initialized) for respective LINKS 30a through 30c. As noted above,
 drivers are loaded according to the type of module involved. For embedded
 systems, the method for installing device drivers will vary depending on the needs
 of the implementation. Device drivers for locally resident drivers may be pre-
 compiled into a ROM image. Under this arrangement, at boot time the drivers
 20 would be called to perform initialization thereof. *a1*
 Link drivers 40a through 40c are
 then configured as described below. ~~TNF kernel 38 becomes aware of LINKS~~
~~30a through 30c through~~ *(please provide*
~~this detail)~~ *1* For each LINK 30a through 30c, the TNF kernel 38 creates link
 device nodes (12a through 12c) using the link data structure 10 as described

above in conjunction with FIG. 2. As noted above, other data structures may be used without departing from the spirit and scope of the present invention.

The TNF kernel 38 then queries each link driver 40a through 40c for the link capabilities of the LINKS 30a through 30c. In the preferred embodiment, the TNF kernel 38 requests each link's capabilities as soon as it becomes aware of the link device. In response, link drivers 40a through ^{40c}40 provides the capabilities of respective LINKS 30a through 30c by providing link capabilities data, which is stored into data structure 24a through 24c for respective nodes 12a through 12c.

The TNF kernel 38 may also receive other link configuration data provided by a user. The TNF kernel 38 evaluates the link capabilities of LINKS 30a through 30c and the other link configuration data, if any, to generate appropriate link driver configurations according each LINK's capabilities and user-defined configuration. The link configuration generated by TNF kernel 38 is then stored to link configuration data structure 22a through 22c associated with respective nodes 12a through 12c in link data structure 10 and is communicated to link drivers 40 a through 40c for configuration therein.

Referring now to FIG. 4., there is shown a functional block diagram of a second exemplary embodiment serial device module 42 which carries out device driver configuration according to the present invention. Module 42, like module 26, includes three nodes 28a through 28c, each having a respective link layer (LINK) device 30a through 30c connected to a respective physical layer (PHY) devices 32a through 32c. LINKS 30a through 30c provide the link services for the module 42, and PHY devices 32a through 32c provide the physical layer

services for the module 42. Each PHY device 32a through 32n is connected to serial bus 34 via conventional serial interface connection.

The module 42 also includes one or more unit architectures 36 to present
 5 to other devices on the serial bus. The unit architectures 36 are operatively coupled for communication to TNF kernel 38. The TNF kernel 38 provides transactional services for module 42 and the link driver configuration of LINKS 28a through 28c as described above and in conjunction below with FIG. 5.

Sub 10 BY
~~The module 42 further includes device driver services (IO coordinator 44)~~
 operatively coupled to the TNF kernel 38 and the LINKS 30a through 30c. The
 IO coordinator 44 provides, among other things, ^{event} even notification to TNF kernel
~~38 of LINKS 30a through 30c.~~

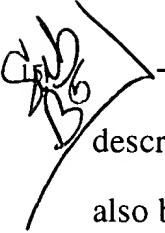
15 In operation, when module 42 is initialized, Link drivers 40a through 40c are loaded (initialized) for respective LINKS 30a through 30c. In response to the Link drivers 40a through 40c being loaded, the IO coordinator 44 communicates to the TNF kernel 38 that Link drivers 40a through 40c have been initialized for LINKS 30a through 30c.

20

For each LINK 30a through 30c, the TNF kernel 38 creates link device nodes (12a through 12c) using the link data structure 10 as described above in conjunction with FIG. 2. As noted above, other data structures may be used without departing from the scope of the present invention.

25

The TNF kernel 38 then queries each link driver 40a through 40c for the link capabilities of the LINKS 30a through 30c. In response, link drivers 40a through 40c provides the capabilities of respective LINKS 30a through 30c by providing link capabilities data, which is stored into data structure 24a through 24c for respective nodes 12a through 12c. The TNF kernel 38 may also receive other link configuration data provided by a user. The TNF kernel 38 evaluates the link capabilities of LINKS 30a through 30c and the other link configuration data, if any, to generate appropriate link driver configurations according each LINK's capabilities and user-defined configuration. The link configuration generated by TNF kernel 38 is then stored to link configuration data structure 22a through 22c associated with respective nodes 12a through 12c in link data structure 10 and is communicated to link drivers 40 a through 40c for configuration therein.

 While the above illustrative embodiments (module 26 and module 42) were described using three nodes 28a through 28c, the method of the invention may also be carried out with a module have one or more nodes. As illustrated, the method of the invention may be carried out with or without device driver services (IO coordinator 44, or other like messaging services).

20

The method and operation of the invention will be more fully understood by reference to the flow chart of FIG. 5, as well as FIG. 2 through FIG. 4. FIG. 5 illustrates generally the actions associated with providing dynamic configuration ROM using double image buffers in accordance with the present invention. The

order of operation as shown in FIG. 4 and described below are only exemplary, and should not be considered limiting.

At box 100, the TNF kernel 38 has become aware of one or more link
5 drivers (and corresponding link devices) which need to be configured according
each respective link device's capabilities, behaviors, as well as other user-defined
configuration data. The TNF kernel 38 may recognize the link devices using
various methods, such as via a device driver notification service, for example, as
described above in conjunction with FIG. 4. For each link device, the TNF kernel
10 38 transmits a "get link capabilities" driver control command to each link driver
initialized for the link devices. As noted above, the TNF kernel maintains the link
device information in a data structure (for example, data structure 10). For each
link device, the TNF kernel creates a data record for recording the capabilities of
the link device. Box 110 is then carried out.

15 At box 110, the link driver receives the "get link capabilities" driver
control command from the TNF kernel 38. As noted above, the capabilities of a
link device may be statically provided in a resident storage device, such as a BIOS
(basic input/output system, for example. The link driver ascertains these
20 capabilities and transmits the capabilities to the TNF kernel 38. Box 120 is then
carried out.

At box 120, the TNF kernel 38 receives the link device's capabilities
transmitted by the link driver from box 110. The link device capabilities may
25 include such information as maximum sync packet capabilities, support or non-

support for busy mode, cycle accuracy, capability for cycle master, CSR-ROM support, and other like capabilities. The TNF kernel 38 may also receive additional configuration data such as user-defined configuration, which may define specific behaviors of the link device. Box 130 is then carried out.

5

At box 130, the TNF kernel 38 evaluates the link device's capabilities and behaviors and provides link driver configuration accordingly. Box 140 is then carried out.

10

At box 140, the TNF kernel 38 communicates the link driver configuration generated from box 130 to the link driver for configuration therein. As noted above, the TNF kernel uses the link data structure 10 and the associated link configuration data structure 22a through 22n to instruct the link driver how it should be configured. The data structure 22a through 22n is used to control how the link behaves on the IEEE Standard 1394 bus, among other things. Box 150 is then carried out.

15

At box 150, the link driver configures itself according the configuration provided by the TNF kernel 28 in the link driver data structures 22a through 22n.

20

Accordingly, it will be seen that this invention provides a method which provides multiple link device driver configurations based on the capabilities of the link device and based on the specific behaviors of the link device. Although the description above contains many specificities, these should not be construed as limiting the scope of the invention but as merely providing an illustration of the

25

presently preferred embodiment of the invention. Thus the scope of this invention should be determined by the appended claims and their legal equivalents.